Unidad 3
Programación Orientada a Objetos. Programación JAVA. Parte I

Nota:

Alumno: _____

1. Corregir los errores del siguiente código JAVA. (0.5p):

```
// Obtener el menor divisible por n proporcionado por el usuario
public class Ejercicio1 {
	public static void main(String[] args) throws IOException  {
		int vector[];
		int numero=0;
		int mayor= Integer.MIN_VALUE;
		int menor= Integer.MAX_VALUE;
		BufferedReader in=new BufferedReader(new
		InputStreamReader(System.in));
		System.out.println("¿Dime la longitus del vector?");
		int longitud=Integer.valueOf(in.readLine()).intValue();
		System.out.println("¿Dime el n divisible?");
		numero=Integer.valueOf(in.readLine());
		vector=new int[longitud];
		for(int i=0 ; i<vector.length ;)
		{
			System.out.println(".....:");
			vector[i] = Integer.valueOf(in.readLine()).intValue();
			if ((vector[i] % numero) == 0) || (vector[i] < menor))
				menor= vector[i];
		}
		System.out.println("....: "+mayor);
		System.out.println("....: "+menor);
	}
}
```

2. Corregir los errores del siguiente código JAVA. (0.5p):

```
import java.io.*;
public class Ejercicio2
{
	public static primo() {
		boolean encontrado = false;
		int divisor=2;
		while(divisor < num && num % divisor == 0)
		{
			divisor++;
		}
		return num != divisor?true:false;
	}
	public static void main(String[] args) throws IOException{
		BufferedReader in=new BufferedReader(new
		InputStreamReader(System.in));
		System.out.println("Introduce un numero: ");
		int num = Integer.valueOf(in.readLine().trim()).intValue();
		if (primo(num))
			System.out.println("El numero "+num+" es primo");
		else
			System.out.println("EL numero "+num+" no es primo");
	}
}
```

3. Qué se añadiría para mejorar -robustez- el código que se proporciona (0.5p):

Atributos de la clase Viaje:

- ✓ Números de trayectos (int). int ntra;
- ✓ Vector de (class Trayecto). Trayecto [] sectrayectos;

Clase Trayectoria

```
public class Trayecto {
        public String origen;
        public String destino;
        public double distancia;
        .....}
```

```
public Trayecto get_trayecto (int p){
        return(this.sectrayectos[p])
}
```

4. Qué se añadiría para mejorar -robustez- el código que se proporciona (0.5p):

```
// Obtener la media de un vector de enteros
public class Ejercicio4 {
    public static void main(String[] args) throws IOException {
                int vector[] = new int[10]
                int menor= Integer.MAX_VALUE;
                int suma=0;
                BufferedReader in=new BufferedReader(new InputStreamReader(System.in));
                for(int i=0 ; i< vector[i]; i++)
                {
                   System.out.println(".....:");
                   vector[i] = Integer.valueOf(in.readLine()).intValue();
                   suma+= vector[i];
                }
                System.out.println("....: "+suma/vector[i]);
   }
}
```

5. Queremos modelar una casa con muchas bombillas mediante la clase Iluminación (vector de Bombillas), de forma que cada bombilla se puede encender o apagar individualmente. Para ello tenemos una clase Bombilla con una variable privada que indique si está encendida o apagada, métodos para pagagar y encendar, un método que nos dice si una bombilla concreta está o no encendida y el constructor que al crear/new un objeto Bombilla la situa es estado de apagada. Tenemos definina una clase Iluminación que necesita los siguientes métodos (0,5p – 0.10/método):

```
class Bombilla
 {
  private ...; // interruptor
  public void enciender () { ... }
  public void apagar () { ... }
  public boolean encendida () { .}
  public Bombilla (){ ... }
  }
```

```
class Iluminacion
{
private ...;  // Bombillas de la casa
private ...;  // Número de bombillas de la casa
public void apagar_bombilla() { ... }
public void encender_bombilla() { ... }
public boolean estado_bombilla() { …}
public numero_bombillas_encendidas (){ ... }
public numero_bombillas_apagadas (){ ... }
public Iluminación { ... }
}
```

6. ¿Qué resuelve el método int z(int num)? (0.5p):

```
import java.io.*;
public class Ejercicio6 {
public static int z(int num){
        int y = 1;
        for (; num>1 ; num--) y = y * num;
        return(a);
}
public static void main(String[] args) throws IOException{
        BufferedReader in = new BufferedReader(new
        InputStreamReader(System.in));
        System.out.println("……..: ");
        int num = Integer.valueOf(in.readLine().trim()).intValue();
        System.out.println("…….: "+z(num));
        }
}
```

7. Escriba en la clase Java Gpoligono el siguiente método (0.5p):

Atributos:

- ✓ Números de lados (*int*). int nlados;
- ✓ Vector de (*class* ***Punto*****)**. Punto [] posicionamiento;
- ✓ Color de relleno (*class* ***Rgb***). Rgb color;

Métodos:

- ✓ public gpoligono(){// Constructor por defecto}
- ✓ public gpoligono(int num, Rgb c){/* Mediante este constructor sobrecargado se proporciona el número de lados y su color. Dentro del cuerpo de este método te preguntará por cada uno de los puntos (X,Y) que formarán su posicionamiento en el espacio.*/}
- ✓ public int getlados(){// Método que devuelve los lados del Gpoligono.}
- ✓ public Rgb getcolor(){// Método que devuelve el color del Gpolígono.}

- ✓ public boolean compareTopoligono(Gpoligono a){/* Método que dado un polígono determina si es igual a él. Son iguales cuando coinciden el color, número de lados y su posicionamiento. */}
- ✓ public Gpoligono mayorTopoligonos(Gpoligono [] a){/* Método que dado un vector de polígonos devuelva el polígono con mayor número de lados. */}

New Método: public int suma_lados(Gpoligono [] a){/* Método que dado un vector de polígonos devuelva la suma de los lados de cada uno de los polígonos. */}

```
public int suma_lados(Gpoligono [] a) {




} // Fin del new método
```

8. Escriba los nuevos métodos de la clase Java que represente un Viaje(0.5):

Atributos:

- ✓ Números de trayectos (int). int ntra;
- ✓ Vector de (class Trayecto). Trayecto [] sectrayectos;

Clase Trayectoria

```
public class Trayecto {
        public String origen;
        public String destino;
        public double distancia;
```

.....}

Métodos:

- ✓ public viaje(){// Constructor por defecto}
- ✓ public viaje(int num){/* Mediante este constructor sobrecargado se proporciona el número de trayectos. Dentro del cuerpo de este método te preguntará cada uno de los trayectos que forman el viaje.*/}
- ✓ public Trayecto mayortrayecto()){/* Devuelve el mayor trayecto en longitug.*/}
- ✓ public double distanciaviaje(){// Resultado de la suma de los trayectos}
- ✓ public boolean combinables(Viaje v1) {/*Si coincide el destino de último trayecto de v1 con el origen del primer trayecto de this, los trayectos son combinables true*/}.
- ✓ public double distancia_combinable(Viaje a){/* Método que dado un viaje devuelve la distancia de la combinación del mismo con this si así fuera posible . */}

New Método: public boolean vcombinables(Viajes [] a){/* Método que dado un vector de viajes devuelve true si son combinables a[0] con a[1], a[1] con a[2] y así sucesivamente. Se devolverá false en otro caso*/}

```
public boolean vcombinables(Viajes [] a)
{




} // Fin del new método
```

9. Escriba los nuevos métodos de la clase Java que represente un ArrayAsir(0.5):

```
Clase ArrayAsir
/* clase que moldea y contiene las operaciones relacionadas con
un vector de enteros (int v[]) */
public class ArrayAsir{
        int [] elementos;
        int num_elemente;
        //métodos
        public int mayor_ArrayAsir(){// devuleve el mayor
```

de los enteros contenidos en la clase}

.....}

Nuevo Método: public int vmayor(ArrayAsir [] a) {/ Devuelve el mayor de todos los enteros contenidos en el vector de ArrayAsir que se recibe como parámetro*/}*

```
public int vmayor(ArrayAsir [] a)  {




} // Fin del new método
```

10. Escriba *uno de los tres métodos que se proponen* de la clase Java que represente un **Dispositivo**:

Se dispone de un dispositivo que es alimentado desde el exterior con instrucciones para que realice determinado tipo de acciones. Se desea modelar el comportamiento de este dispositivo en Java. Para ello, se utilizarán las clases Instrucción y Dispositivo que se detallan a continuación.

```
public class Instruccion{
        private String codigo; // nombre de la instrucción
        public int prioridad; // prioridad de la instrucción
        public long tiempo; // instante en el que se invoca la instrucción
```

```
        // Contructor de inicializar cada uno de los atributos de una instrución
        public Instruccion (String codigo, int prioridad, long tiempo) { ... }
       // Devuelve el código/nombre de la insrtucción. Por ejemplo podría devolver
       "SUM"
        public String getCodigo () {…. }
       // Método que compara dos instrucciones devolviendo true si ambas tienen el
       mismo código/nombre de acción
       public boolean compareToIntruccion (Instrucción aux){…}
}

public class Dispositivo{
        private Instruccion[] cola; // vector de instrucciones
        private int numInstrucciones; // número de instrucciones en el dispositivo
       // Constructor tamaño inicial de 100
       public Dispositivo () {
            this.cola=new Instrucción[100];
            this.numInstrucciones=-1;          }
       // Constructor con tamaño inicial proporcionado por el usuario
       public Dispositivo (int tama_max) { …};
        // Método que se ejecuta cuando se quiere introducir una instrucción
        public void putInstruccion (Instruccion i) { ... }
       // Método que determina cuantas instrucciones son del nombre/código "XX"
        public int count_codigo (String codigo) { ... }
       // Método que ejecuta una instrucción.
       public void ejecutarInstruccion(){….}
}
```

- *Nuevo Método (0.5p): public int pos_mayor prioridad () {/* Devuelve la posición de la instrucción con mayor prioridad. Si el sistema está vacío devuelve -1}*

- *Nuevo Método (0.5p): public void ejecutarInstruccion(int pos){ /*Método que ejecuta la instrucción que ocupa la posición p. La ejecución de la instrucción cola[p] implica simularlo mostrando por pantalla las instrucción y a continuación desplazando todas la siguientes una posición a la izquierda y actualizando el valor de numInstrucciones. Si no hay instrucciones debe aparecer por pantalla "dispositivo vacío" */}*

- *Nuevo Método (0.5p): public Dispositivo mayor_carga (Dispositivo [] aux) {/*Método que devuelve el dispositivo con mayor número de instrucciones a ejecutar */}*

XXXX {

} // Fin del new método

11. Cuestionario tipo test(2 mal quitan 1 bien): (2 puntos)

1) ¿ Qué ocurre cuando se compila y ejecuta la siguiente clase ?

```
class MiClase {
public static void main (String[] args) {
String s1[] = new String[5];
String str = s1[0].toUpperCase();
System.out.println(str);
}
}
```

   a. Imprime NULL.
   b. Da un error al compilar.
   c. Imprime null.
   d. Da una excepción NullPointerException al ejecutar.

2) Qué se escribe por pantalla con la siguiente línea de código: System.out.println ((int) Math.PI);
   a. 3.1415926
   b. 3
   c. nada, hay un error al compilar.
   d. nada, hay un error durante la ejecución.

3) Dada la siguiente definición: ¿Cuál debería ser el contenido del constructor?

```
class Temperatura {
double t;
Temperatura(double t) { ¿? }
}
```

   a. t=t;
   b. double t=t;
   c. this.t=t;
   d. No se puede llamar igual el parámetro del constructor que el atributo de la clase

4) "for (x= 5; x <100; x*= 2) { cosas; }" es equivalente a ...
   a. x= 5; do { cosas; x*= 2; } while (x <100);
   b. x= 5; while (x <100) { cosas; x*= 2; }

5) 2. Dado el siguiente fragmento de programa: Indique que afirmación es cierta:

```
int k;
for (k=5 ; k>0 ; k--)
System.out.print(k);
System.out.print(k);
```

a. Se imprime 543210
b. Se imprime 5432100
c. Se imprime 554433221100
d. Se imprime 543210-1

6) Indique el elemento que no es obligatorio en una declaración de variable:

a. La asignación del valor inicial.
b. El identificador o nombre.
c. El punto y coma.
d. El tipo.

7) Indique la salida de:

```
int a= 7, b= 3;
System.out.println ((++a) * b);
```

a. 24
b. 21
c. 10
d. 73

8) Evalúe el valor final que toma la variable "s":

```
int n= 1; s= 0;
while (n <= 9) s+=n;
```

a. 45
b. 0
c. 9
d. el programa no termina nunca

9) ¿ Qué imprime este programa ?

```
int metodo (int v1) { return v1*v1; }
  .........
int v = 3;
System.out.print (metodo (metodo (v))); ...
```

a. Tiene errores que impiden su compilación
b. 9
c. 81
d. Compila; pero tiene errores que impiden su ejecución

10) ¿Qué se imprimirá al ejecutar el siguiente bucle?

```
for (int i=0; i < 5; i++) {
if (i==3) { i=5; }
System.out.println (i + " ");
```

```
}
```

a. 0 1 2 3 4
b. 0 1 2 5
c. 0 1 2 4
d. 0 1 2

11) Indique qué ocurre con el siguiente código:

```
int a[] = new int[10];
for (int i=0 ; i<=a.length ; i++)
  System.out.print(a[i]);
```

a. Error de ejecución: índice fuera de rango.
b. Error de compilación: no se asignaron valores al array.
c. Escribe los 10 valores almacenados en a.
d. Escribe 10 ceros.

12) Indique qué escribe una llamada al método m()

```
void m() {
  int x = 0;
  try {
    System.out.print(x++);
    if (x>0) throw new Exception();
    System.out.print(x++);
  } catch (Exception e) {
    System.out.print(x++);
    } finally {
    System.out.print(x++);
  }
}
```

a. 012
b. 0
c. 01
d. 0123

13) Dados los siguientes fragmentos de código:

```
class ClaseC {
public void fmet (int i) { ... }
public int fmet (int i) { ... }
}  ...
ClaseC c = new ClaseC();
c.fmet(4);
```

Se produce:

a. la llamada al primer método fmet.
b. la llamada a ningún método porque hay sobrecarga.
c. un error al compilar.
d. un error al ejecutar.

Dado el método "imprime":

```
void imprime (boolean ok) {
 System.out.println ("La respuesta " + (ok?"NO":"") + " está mal");
```

```
}
```

¿qué imprime la llamada imprime(false)?

a. La respuesta está mal
b. La respuesta NO está mal
c. Se produce un error de compilación

14) En una sentencia try-catch-finally:

a. Los bloques catch se pueden repetir tantas veces como excepciones de distinto tipo se desee atrapar. El bloque finally debe aparecer al menos una vez y se ejecuta siempre.
b. Los bloques catch se pueden repetir tantas veces como excepciones de distinto tipo se desee atrapar. El bloque finally no es opcional y se ejecuta siempre.
c. Los bloques catch se pueden repetir tantas veces como excepciones de distinto tipo se desee atrapar. El bloque finally es opcional y solo puede aparecer una vez. Este bloque se ejecuta siempre.
d. Todas las afirmaciones son falsas

Unidad 3
Programación Orientada a Objetos. Programación JAVA. Parte II

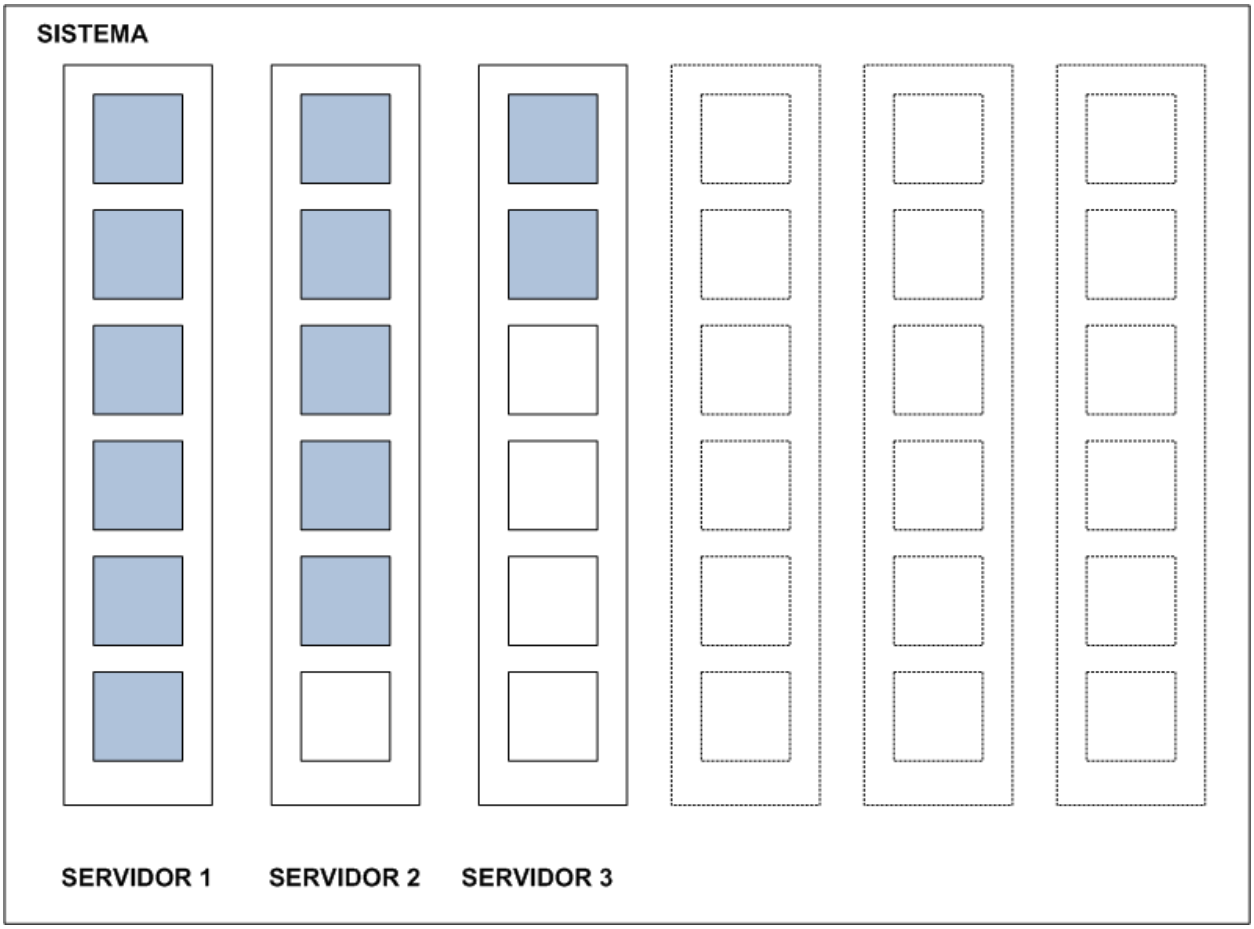
12. Desarrolla en java el Sistema de Información CajeroAutomático: (1,5p)

```
public class Operacion
{
        private String nTarjeta; // número de la tarjeta
       public String operación; // tipo de operación por ejemplo VERSALDO
       public int cantidad; // Cantidad monetaria retirada en una operación de RETIRAR
        // Contructor de inicializar cada uno de los atributos de una instrución
        public Operacion (String nt, String op) { ... } // En este caso la cantidad tendrá un
       valor de -1 euros
       public Operacion (String nt, String op, int cantidad) { ... }
       // Devuelve el código de la operación.
        public String getOperacion () {…. }
        }

public class CajeroAutomatico
{
        public Operacion[] cola; // vector de operaciones
        private int numTransacciones; // número de instrucciones en el dispositivo
       // Constructor tamaño inicial de 100
       public CajeroAutomatico () {
            this.cola=new Operacion[100];
            this. numTransacciones =0;          }
       // Constructor con tamaño inicial proporcionado por el usuario
       public CajeroAutomatico (int tama_max) { …};
        // Método que incorpora una operación al sistema
        public void putITransacciones (Operacion i) { ... }
       // Método que determina cuantas operaciones ha realizado una trajeta
        public int count_operaciones(String nt) { ... }
       // Método que dado una tarjeta determina la cantidad total de dinero retirada.
       public int cantidadRetirado(){….}
}
```

13. Desarrolla en java este Sistema de Información: (1,5p)

Se dispone de un sistema informático *Sistema.java* capaz de cargar y procesar tareas genéricas *Tarea.java*; estas tareas son procesadas en servidores internos del sistema *Servidor.java*. Estos servidores disponen de colas de tamaño limitado, y es en estas colas donde se van almacenando las tareas que entran en el sistema.
El sistema va creando servidores a medida que los va necesitando, hasta llegar a un límite predeterminado.



Cuando una tarea llega al sistema (vector de servidores), esta se añade a la cola (vector) de proceso de uno de los servidores de acuerdo con las siguientes reglas:

1. Si alguno de los servidores tiene capacidad para admitir tareas (si hay sitio en su cola), se añade en ese servidor.
2. Si se ha alcanzado el límite de servidores posibles, la tarea se pierde.
3. Si no, el sistema crea un nuevo servidor y carga la tarea en la cola de ese servidor.

Se pide:

- Definir los atributos necesarios y el constructor de cada una de las clases necesarias para implementar el sistema.
- Escribir el método putTarea() de la clase Servidor
- Escribir el método getTarea() de la clase Servidor
- Escribir el método cargaTarea() de la clase Sistema